

```

#ifdef _EXITTER
#define _EXITTER_
#include "l3fchunk/L3NodeStat.hpp"

#ifdef ERRORLOG_H
#include "ErrorLogger/ErrorLog.h"
#endif
#include "edm/Event.hpp"
#include "l2base/types.hpp"
#include "l2base/consts.hpp"
#include "itc_event/Event_Message.hpp"
#include "l3fcoor/ClientDef.hpp"

#ifdef _RUN_CFG_MGR_
#include "run_config_mgr/run_cfg_mgr.h"
#endif

#include "l3fchunk/L3MonChunk.hpp"
//Class Externlterator
//
// Purpose: This class (friend class of the List class, as usual) provides
// loop-like methods to traverse the "external Tree" and, for each L2 bit step,
// a internal traversing is made.
// Will provide all the functionalities of calling Tools & Filters "reset"
// methods, as well as collecting statistic informations about Filters
// and Tools
//
// Created 12/12/97 Moacyr Souza
//
// Modifications : 12/22/98 : merged with previous Internlterator class
// (Moacyr Souza 12/12/97)
//
//
//
typedef char * bufferType;
typedef std::map<const char *, NodeStat> L3StatMap;
typedef std::map<std::string, LumStat> LumMap;

//forward declarations
class bTool;
class List;
class LinkNode;
class L3MonChunk;
class llfw_info;
class l2tfw_info;

class Externlterator {
public:
//Constructor
Externlterator ();
//Destructor
~Externlterator();

// this is the "return" status for one event
bool Passed();
// this is the debug flag
bool Debug_On();

// Method "Traverse"
// To traverse a given L2 trigger bit branch.
bool Traverse();

// loop-like methods/operators
void Begin();
bool isFinish() const;
void operator ++();

```

```

linkNode *_Current() const;

//resets
void ResetTools();
void ResetFilters() const;
void ResetMySelf();

//Tools run_config RunInit
void ToolsRunInit();

//chunk generators
bool L3ChunkGen(edm::Event *);
bool L3DebugChunkGen(edm::Event *);

//hook servicing methods
bool BeginRun( std::bitset<l2base::MAX_NUM_L2_BITS> & );
bool EndRun( std::bitset<l2base::MAX_NUM_L2_BITS> & );
void Summary();
// this will return the event header;
_ITC_EVENT_NAMESPACE::Event_Header *getEventHeader();

// This method parses the Trigger List on duty. For each line, actions
// are taken to instantiate Tools and Filters and methods are called
// to build the execution Tree, on the fly. Called from the ScriptRunner
// constructor and takes a List name as argument.
// Created: 05/05/98 Carmem Silva
bool AddL2bits( l3string & );

void DropL2Bits( std::bitset<l2base::MAX_NUM_L2_BITS> & );
void DropL2MonInfo(std::bitset<l2base::MAX_NUM_L2_BITS> & );

bool isThere();
// this will (hopefully) frees all allocated memory on the heap
void Destroy();

//this will set up a "daq_test" bank
void setupTest();

// time
clock_t ElapsedTime() const;
void uptime(clock_t );

void UpdateL2Map(l3map<int, l3string> &_l2tmp );
//"same" for L3 Mat
void UpdateL3Map( l3map<int, ClientDef * > & );

// generate the Event Header
bool genEventHeader(const edm::Event&);

//update L2 bits for ITC header
void ITC_L2Bits();

// Monitoring
bool getMonData( L3MonChunk & );
bool getLumData();
bool getLumData( L3MonChunk & );
void setMonMap(); // this will erase _l2run_map
void setMonMap(int, const std::bitset<l2base::MAX_NUM_L2_BITS> & );
int getMonId();

//get L1/L2 info
bool getL1L2Info(const edm::Event &);

//get the lum_index
unsigned int GetLumIndex();
void setCrateList(const l3vector<int> &crateList);

```

```

//return online/offline
bool Is_Online();
void Set_Online(bool );

private:
edm::Event *event;
//online/offline
bool_online;

// this event did pass
bool_passed;
// write debug info
bool_writeDebug;
// write debug info each _debugInfo'th
int_debugInfo;
//ship complete monitor info each _monitorInfo'th
int_monitorInfo;
bool_monFull;

// flags a calibration run
bool_calib_on;

//flag to use L1 bits
bool_useL1;
bool_useL2;
List *triglist;

clock_t elapsedTime;
// this is the external loop index
int_extIndex;
// this is the total # of L2 bits parsed and on duty *now*
int_extIndexMax;

//this is the event header
_ITC_EVENT_NAMESPACE::Event_Header *event_header;

unsigned int evcount;
unsigned int lumcount;

//this is the event nb
unsigned int L1EventNb;

// this is the event mask
std::bitset<L2base::MAX_NUM_L2_BITS> L2Eventbits;
// this is the L2 currently active bits
std::bitset<L2base::MAX_NUM_L2_BITS> L2bits;
// this is the reference counting for L2bits
l3vector<int> _l2ref;
// this is the array of L2bits
int bitsort[L2base::MAX_NUM_L2_BITS];

// this is the L2 bit info for the itc_header
unsigned int *l2itc;
unsigned int l2count;

//same for l3 bits
unsigned int *l3itc;
unsigned int l3count;

// container classes
// run statistics, by l2bit
l3map<int, l3string> *_l2map;
l3map<l3string, int> *_l3map;

```

```

// monitoring statistics
std::map<int, l3StatMap> SumStat;

// monitoring map - runnb<->L2bit
l3map<int, std::bitset<L2base::MAX_NUM_L2_BITS> > _l2run_map;

// Those are traversing pointers
linkNode *_excurrent;
linkNode *_current;
linkNode *_start;

// error logging
static ErrorLog _IteratorLog;
bool L2Unbiased;
//Flag for L2 unbiased events
// Run Configuration Manager:
DOMCH::run_cfg *_runcfg;
int_crateID;

l1tfw_info *l1info;
l2tfw_info *l2info;

bool_endrun ;
//monitoring flag : 0=regular, 1=full, 2=new lumIndex, 3=endrun summary
int_monID;

bool_send_mon_info;
bool_ITCunbiased;

//luminosity index
unsigned int LumIndex;
unsigned int _oldlum;
bool_new_lum;
bool_old_lum;
l3map<int, LumMap> _lumStat;
l3map<int, LumMap> _lumStat_old;
l3map<int, LumMap> _lumStat_old_save;
l3vector<int> _crateList;
};

//-----inlines
inline bool ExternIterator::passed() {
    return _passed;
}

inline clock_t ExternIterator::ElapsedTime() const {
    return elapsedTime;
}

inline void ExternIterator::upTime(clock_t tt) {
    elapsedTime = elapsedTime +tt;
}

inline _ITC_EVENT_NAMESPACE::Event_Header *ExternIterator::getEventHeader() {
    return event_header;
}

inline unsigned int ExternIterator::GetLumIndex() {
    return LumIndex;
}

inline void ExternIterator::setCrateList(const l3vector<int> &crateList)
{
    _crateList = crateList;
}

```

```
Dec 03, 01 0:22      ExternIterator.hpp      Page 5/5  
}  
inline bool ExternIterator::Is_Online() {  
    return _online;  
}  
#endif
```